http://algs4.cs.princeton.edu

4.4 SHORTEST PATHS

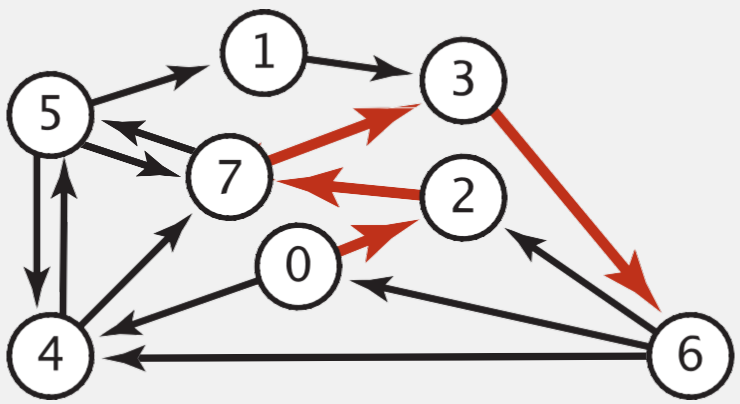
- *APIs*
- *shortest-paths properties*
- *Dijkstra's algorithm*
- *edge-weighted DAGs*
- *negative weights*

Shortest paths in an edge-weighted digraph

Given an edge-weighted digraph, find the shortest path from s to t.

edge-weighted digraph

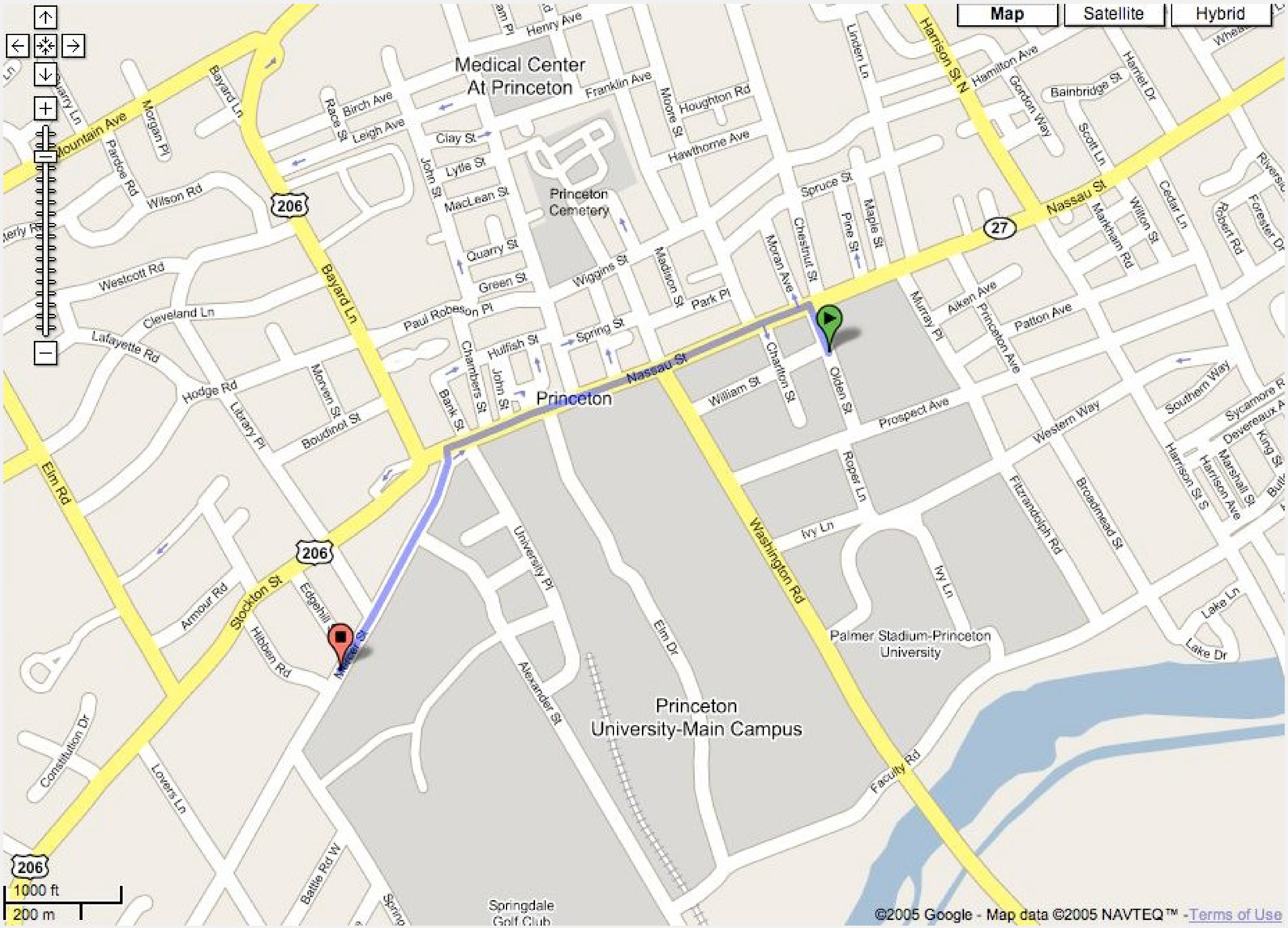
4->5	0.35
5->4	0.35
4->7	0.37
5->7	0.28
7->5	0.28
5->1	0.32
0->4	0.38
0->2	0.26
7->3	0.39
1->3	0.29
2->7	0.34
6->2	0.40
3->6	0.52
6->0	0.58
6->4	0.93



shortest path from 0 to 6

0->2	0.26
2->7	0.34
7->3	0.39
3->6	0.52

Google maps


Map
Satellite
Hybrid
Medical Center
At Princeton
Henry Ave
Franklin Ave
Houghton Rd
Hawthorne Ave
Moore St
Birch Ave
Leigh Ave
Race St
Clay St
Lytle St
MacLean St
John St
Quarry St
Green St
Princeton
Cemetery
Wiggins St
Madison St
Park Pl
Paul Robeson Pl
Hulfish St
Spring St
Chambers St
Bank St
Nassau St
Princeton
William St
Charlton St
Olden St
Prospect Ave
Roper Ln
Ivy Ln
Moran Ave
Chestnut St
Spruce St
Pine St
Maple St
Linden Ln
Harrison St N
Hamilton Ave
Gordon Way
Bainbridge St
Harriet Dr
Scott Ln
Markham Rd
Walton St
Cedar Ln
Robert Rd
Forester Dr
Murray Pl
Aiken Ave
Princeton Ave
Patton Ave
Western Way
Southern Way
Fitzrandolph Rd
Broadmead St
Harrison St S
Marshall St
King St
Lake Ln
Lake Dr
Mountain Ave
Morgan Pl
Pardoe Rd
Wilson Rd
Bayard Ln
Westcott Rd
Cleveland Ln
Lafayette Rd
Hodge Rd
Library Pl
Morven St
Boudinot St
Elm Rd
Armour Rd
Stockton St
Edgehill
Hibben Rd
Mercer St
Constitution Dr
Lovers Ln
Battle Rd W
University Pl
Alexander St
Elm Dr
Washington Rd
Palmer Stadium-Princeton
University
Princeton
University-Main Campus
Faculty Rd
Springdale
Golf Club
206
27
1000 ft
200 m
©2005 Google - Map data ©2005 NAVTEQ™ - Terms of Use

Shortest path applications

http://en.wikipedia.org/wiki/Seam_carving

- PERT/CPM.
- Map routing.
- Seam carving.
- Texture mapping.
- Robot navigation.
- Typesetting in TeX.
- Urban traffic planning.
- Optimal pipelining of VLSI chip.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Exploiting arbitrage opportunities in currency exchange.
- Optimal truck routing through given traffic congestion pattern.

Reference: Network Flows: Theory, Algorithms, and Applications, R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Prentice Hall, 1993.

Shortest path variantat

Cilat kulme?

- Single source: prej një kulmi s tek çdo kulm tjetër.
- Single sink: prej çdo kulmi tek një kulm t.
- Source-sink: prej një kulmi s tek tjetri t.
- All pairs: Në mes të gjitha çifteve të kulmeve.

Restrikimet për peshën e segmenteve?

- Peshat jonegative.
- Peshat Euclidiane.
- Peshat arbitrare.

Ciklet?

- Pa directed cikle.
- Pa "cikle negative."

which variant?

Simplifying assumption. Shortest paths prej s tek çdo kulm v ekzistojnë.

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

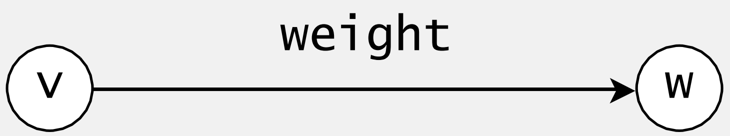
4.4 Shortest Paths

- *APIs*
- *shortest-paths properties*
- *Dijkstra's algorithm*
- *edge-weighted DAGs*
- *negative weights*

Weighted directed edge API

`public class DirectedEdge`

	`DirectedEdge(int v, int w, double weight)`	*weighted edge v→w*
`int`	`from()`	*vertex v*
`int`	`to()`	*vertex w*
`double`	`weight()`	*weight of this edge*
`String`	`toString()`	*string representation*



Idiom for processing an edge `e`: `int v = e.from(), w = e.to();`

Weighted directed edge: implementation in Java

Similar to `Edge` for undirected graphs, but a bit simpler.

```
public class DirectedEdge
{
   private final int v, w;
   private final double weight;

   public DirectedEdge(int v, int w, double weight)
   {
      this.v = v;
      this.w = w;
      this.weight = weight;
   }

   public int from()
   {  return v;  }

   public int to()
   {  return w;  }

   public int weight()
   {  return weight; }
}
```

from() and to() replace either() and other()

Edge-weighted digraph API

public class	EdgeWeightedDigraph	
	EdgeWeightedDigraph(int V)	*edge-weighted digraph with V vertices*
	EdgeWeightedDigraph(In in)	*edge-weighted digraph from input stream*
void	addEdge(DirectedEdge e)	*add weighted directed edge e*
Iterable<DirectedEdge>	adj(int v)	*edges pointing from v*
int	V()	*number of vertices*
int	E()	*number of edges*
Iterable<DirectedEdge>	edges()	*all edges*
String	toString()	*string representation*

Conventions. Allow self-loops and parallel edges.

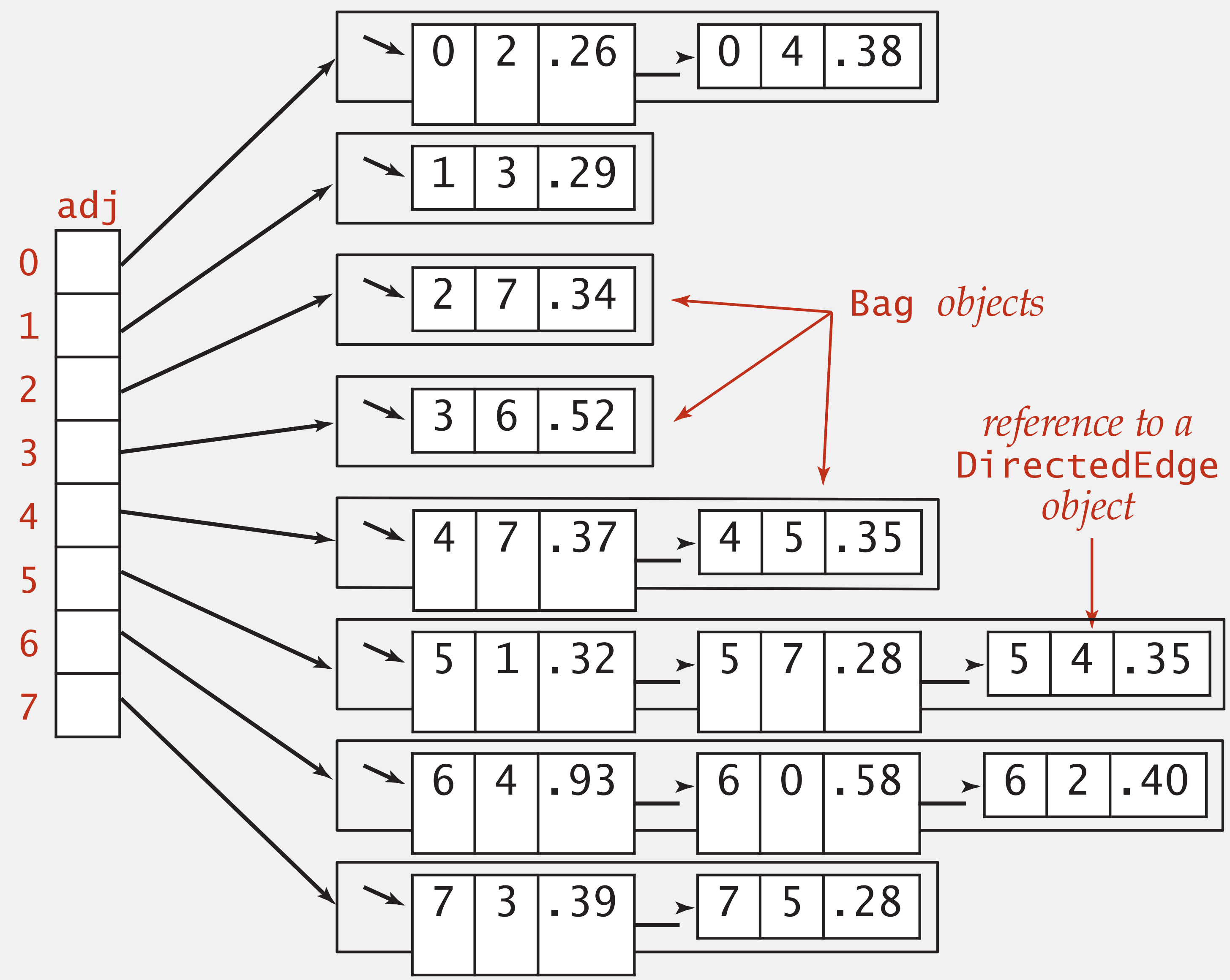
Edge-weighted digraph: adjacency-lists representation

tinyEWD.txt

V → 8

E → 15

```
8
15
4 5  0.35
5 4  0.35
4 7  0.37
5 7  0.28
7 5  0.28
5 1  0.32
0 4  0.38
0 2  0.26
7 3  0.39
1 3  0.29
2 7  0.34
6 2  0.40
3 6  0.52
6 0  0.58
6 4  0.93
```



Edge-weighted digraph: adjacency-lists implementation in Java

Same as `EdgeWeightedGraph` except replace `Graph` with `Digraph`.

```
public class EdgeWeightedDigraph
{
   private final int V;
   private final Bag<DirectedEdge>[] adj;

   public EdgeWeightedDigraph(int V)
   {
      this.V = V;
      adj = (Bag<DirectedEdge>[]) new Bag[V];
      for (int v = 0; v < V; v++)
         adj[v] = new Bag<DirectedEdge>();
   }

   public void addEdge(DirectedEdge e)
   {
      int v = e.from();
      adj[v].add(e);
   }

   public Iterable<DirectedEdge> adj(int v)
   {  return adj[v];  }
}
```

add edge e = v→w to only v's adjacency list

Single-source shortest paths API

Goal. Find the shortest path from s to every other vertex.

public class SP

	SP(EdgeWeightedDigraph G, int s)	*shortest paths from s in graph G*
double	distTo(int v)	*length of shortest path from s to v*
Iterable <DirectedEdge>	pathTo(int v)	*shortest path from s to v*
boolean	hasPathTo(int v)	*is there a path from s to v?*

```
SP sp = new SP(G, s);
for (int v = 0; v < G.V(); v++)
{
   StdOut.printf("%d to %d (%.2f): ", s, v, sp.distTo(v));
   for (DirectedEdge e : sp.pathTo(v))
      StdOut.print(e + "  ");
   StdOut.println();
}
```

Single-source shortest paths API

Goal. Find the shortest path from s to every other vertex.

public class SP

	SP(EdgeWeightedDigraph G, int s)	*shortest paths from s in graph G*
double	distTo(int v)	*length of shortest path from s to v*
Iterable <DirectedEdge>	pathTo(int v)	*shortest path from s to v*
boolean	hasPathTo(int v)	*is there a path from s to v?*

```
% java SP tinyEWD.txt 0
0 to 0 (0.00):
0 to 1 (1.05): 0->4 0.38  4->5 0.35  5->1 0.32
0 to 2 (0.26): 0->2 0.26
0 to 3 (0.99): 0->2 0.26  2->7 0.34  7->3 0.39
0 to 4 (0.38): 0->4 0.38
0 to 5 (0.73): 0->4 0.38  4->5 0.35
0 to 6 (1.51): 0->2 0.26  2->7 0.34  7->3 0.39  3->6 0.52
0 to 7 (0.60): 0->2 0.26  2->7 0.34
```

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

4.4 SHORTEST PATHS

- APIs
- *shortest-paths properties*
- Dijkstra's algorithm
- edge-weighted DAGs
- negative weights

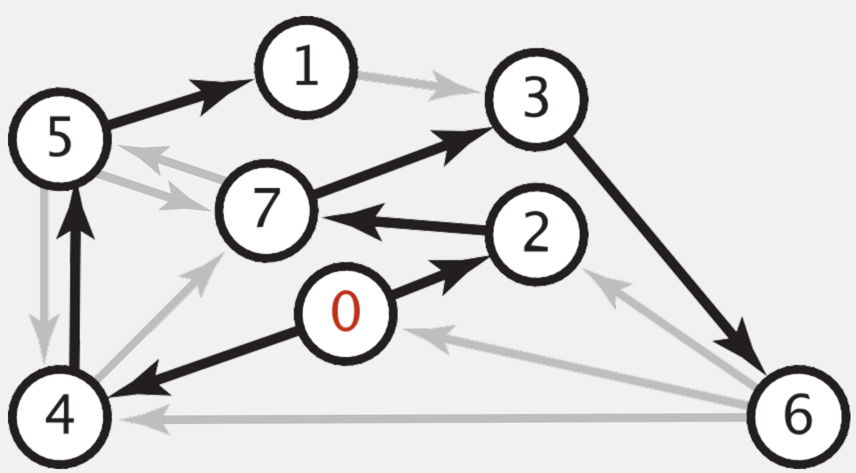
Strukturat e të dhënave për single-source shortest paths

Qëllimi. Gjej shortest path prej *s* tek çdo kulm tjetër.

Vërejtje. shortest-paths tree (SPT) ekziston. Pse?

Përfundim. SPT mund të paraqitet me anë të dy arrays me kulme të indeksuara:

- `distTo[v]` është gjatësia e shortest path from *s* tek *v*.
- `edgeTo[v]` është segmenti i fundit në shortest path prej *s* tek *v*.



shortest-paths tree from 0

	edgeTo[]		distTo[]
0	null		0
1	5->1	0.32	1.05
2	0->2	0.26	0.26
3	7->3	0.37	0.97
4	0->4	0.38	0.38
5	4->5	0.35	0.73
6	3->6	0.52	1.49
7	2->7	0.34	0.60

parent-link representation

Data structures for single-source shortest paths

Qëllimi. Gjej shortest path prej s tek çdo kulm tjetër.

Observation. shortest-paths tree (SPT) ekziston. Pse?

Përfundim. SPT mund të paraqitet me anë të dy arrays me kulme të indeksuara:

- `distTo[v]` është gjatësia e shortest path from s tek v.
- `edgeTo[v]` është segmenti i fundit në shortest path prej s tek v.

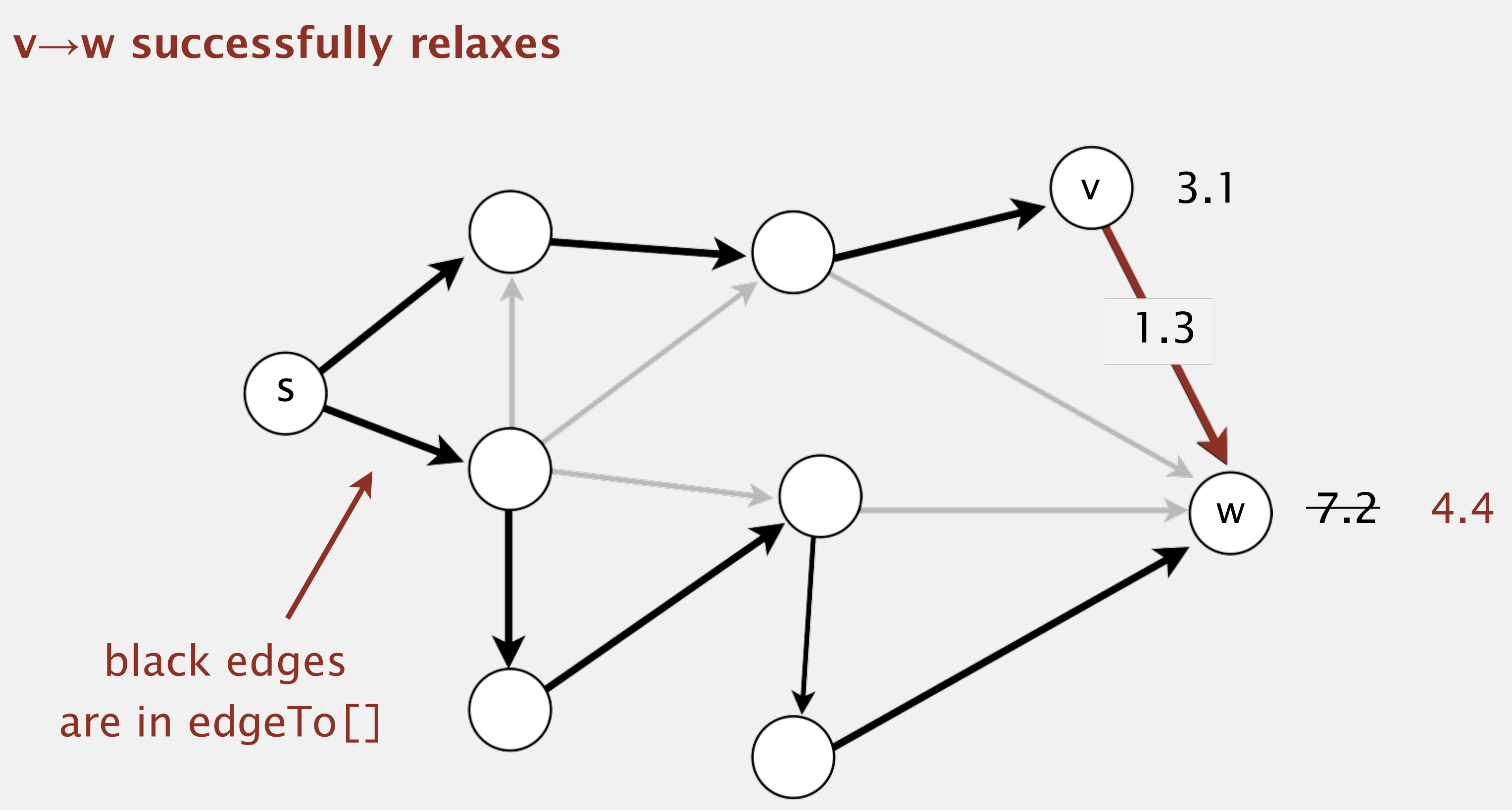
```
public double distTo(int v)
{  return distTo[v];  }

public Iterable<DirectedEdge> pathTo(int v)
{
   Stack<DirectedEdge> path = new Stack<DirectedEdge>();
   for (DirectedEdge e = edgeTo[v]; e != null; e = edgeTo[e.from()])
      path.push(e);
   return path;
}
```

Lehtësimi i segmentit

Lehtësimi i segmentit $e = v \rightarrow w$.

- `distTo[v]` is length of shortest known path from `s` to `v`.
- `distTo[w]` is length of shortest known path from `s` to `w`.
- `edgeTo[w]` is last edge on shortest known path from `s` to `w`.
- If `e = v→w` gives shorter path to `w` through `v`, update both `distTo[w]` and `edgeTo[w]`.



Edge relaxation

Relax edge $e = v \to w$.

- distTo[v] is length of shortest known path from s to v.
- distTo[w] is length of shortest known path from s to w.
- edgeTo[w] is last edge on shortest known path from s to w.
- If e = v→w gives shorter path to w through v, update both distTo[w] and edgeTo[w].

```
private void relax(DirectedEdge e)
{
   int v = e.from(), w = e.to();
   if (distTo[w] > distTo[v] + e.weight())
   {
       distTo[w] = distTo[v] + e.weight();
       edgeTo[w] = e;
   }
}
```

Generic shortest-paths algorithm

Generic algorithm (to compute SPT from s)

Initialize distTo[s] = 0 and distTo[v] = ∞ for all other vertices.

Repeat until optimality conditions are satisfied:

- **Relax any edge.**

Efficient implementations. How to choose which edge to relax?

Ex 1. Dijkstra's algorithm (nonnegative weights).

Ex 2. Topological sort algorithm (no directed cycles).

Ex 3. Bellman-Ford algorithm (no negative cycles).

http://algs4.cs.princeton.edu

4.4 Shortest Paths

- APIs
- shortest-paths properties
- *Dijkstra's algorithm*
- edge-weighted DAGs
- negative weights

Edsger W. Dijkstra: select quotes

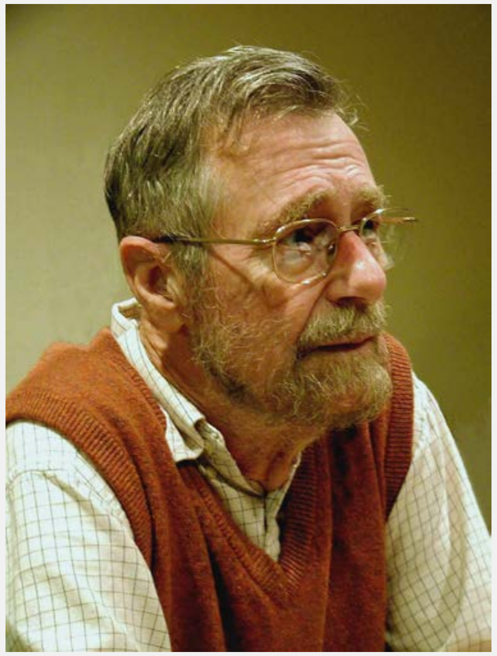
“ *Do only what only you can do.* ”

“ *In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind.* ”

“ *The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.* ”

“ *It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration.* ”

“ *APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.* ”

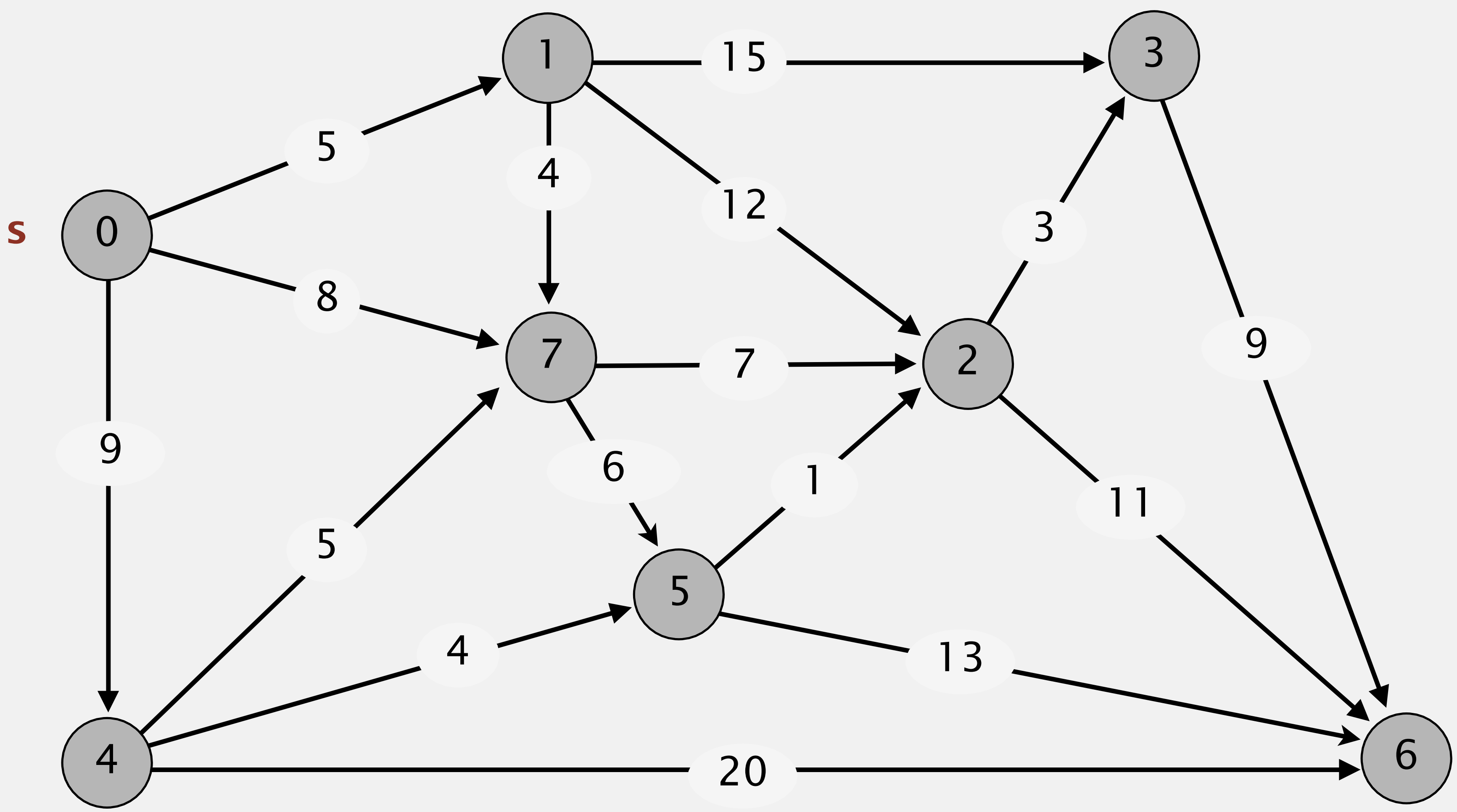


Edsger W. Dijkstra
Turing award 1972

Edsger W. Dijkstra: select quotes

Dijkstra's algorithm demo

- Le të jenë kulmet duke u rritur për nga distanca prej `s` (non-tree vertex with the lowest `distTo[]` value).
- Shto kulmin në dru dhe lehtëso të gjitha segmentet që dalin nga ai kulm.

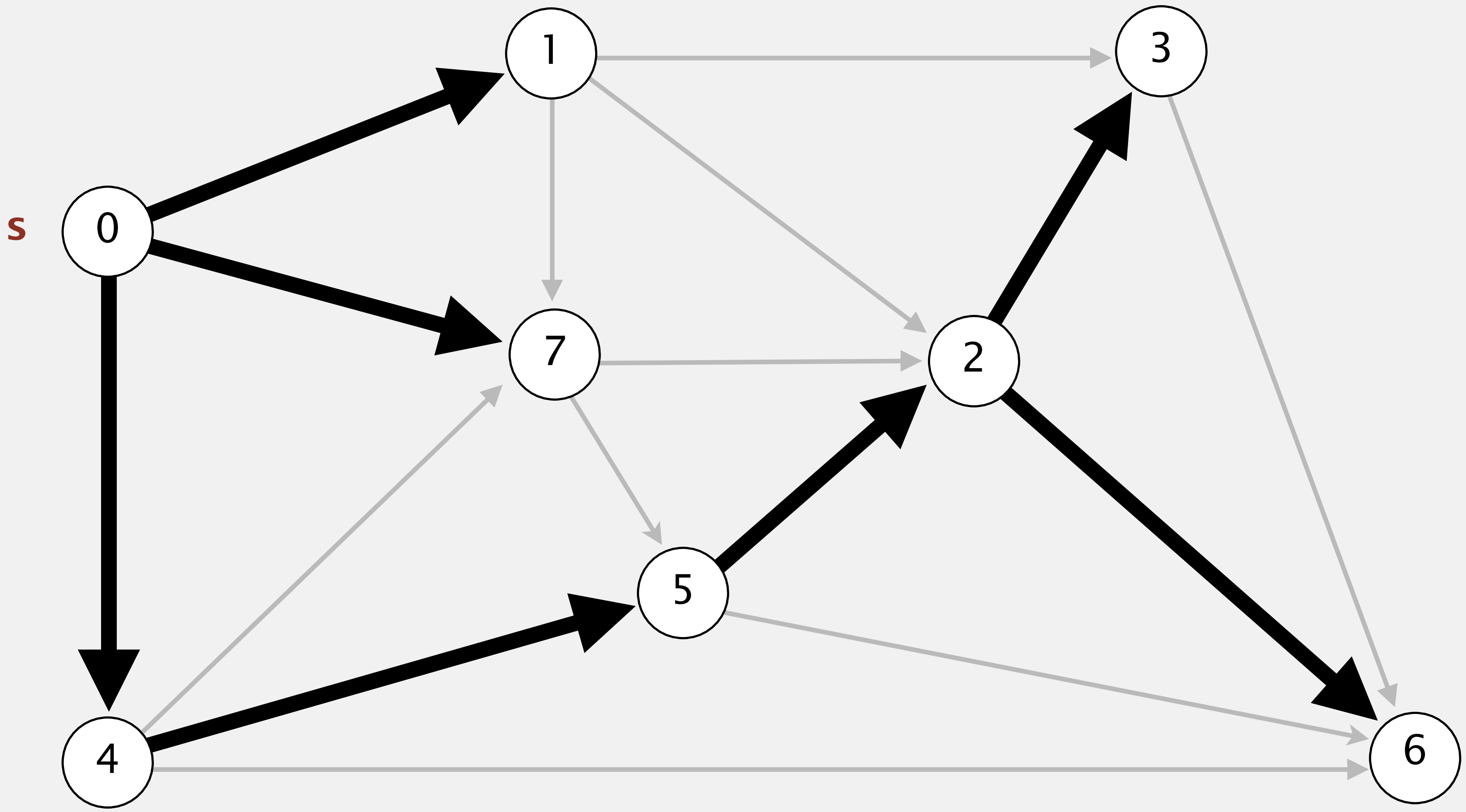


an edge-weighted digraph

0→1	5.0
0→4	9.0
0→7	8.0
1→2	12.0
1→3	15.0
1→7	4.0
2→3	3.0
2→6	11.0
3→6	9.0
4→5	4.0
4→6	20.0
4→7	5.0
5→2	1.0
5→6	13.0
7→5	6.0
7→2	7.0

Dijkstra's algorithm demo

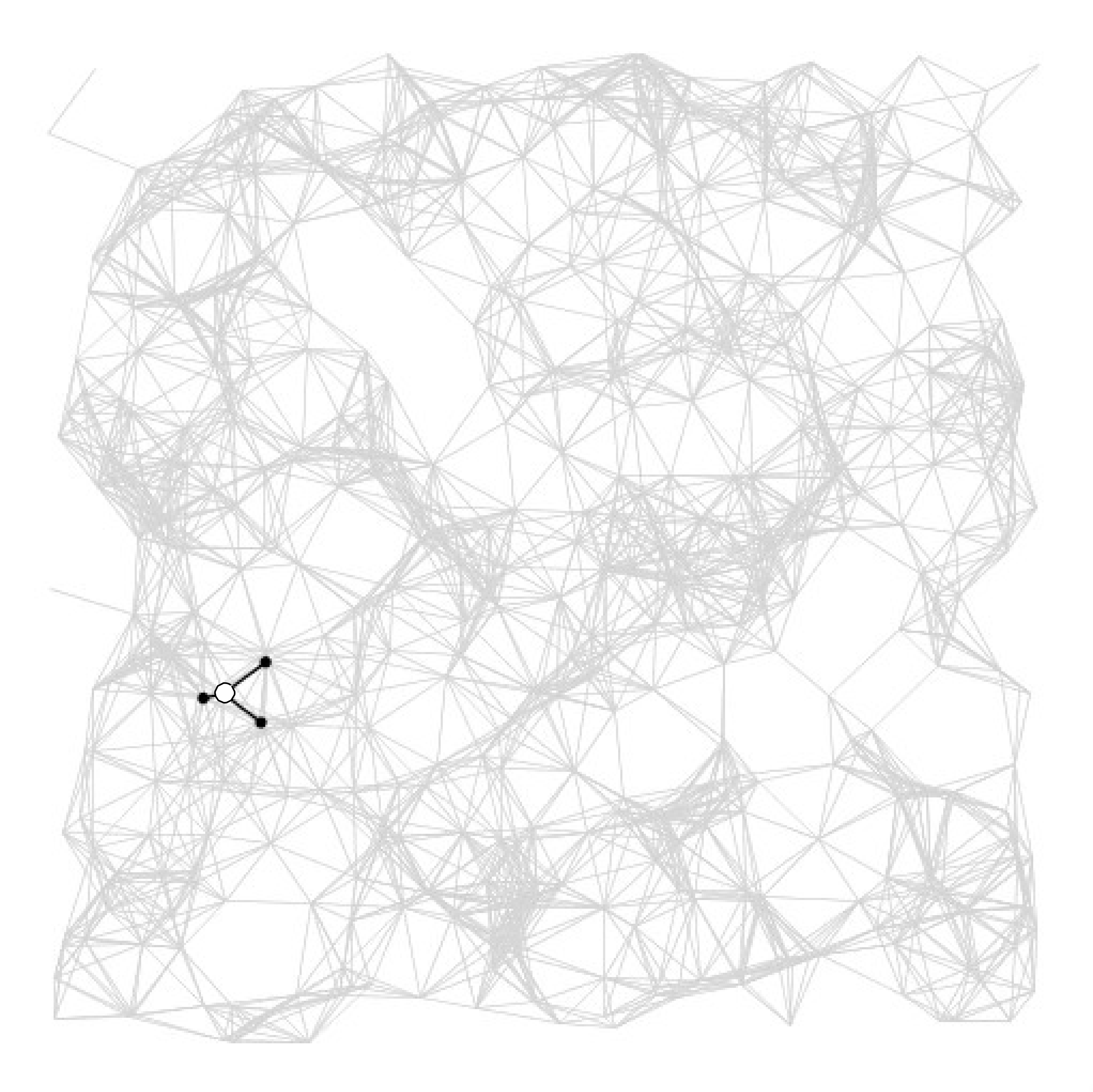
- Le të jenë kulmet duke u rritur për nga distanca prej `s` (non-tree vertex with the lowest `distTo[]` value).
- Shto kulmin në dru dhe lehtëso të gjitha segmentet që dalin nga ai kulm.



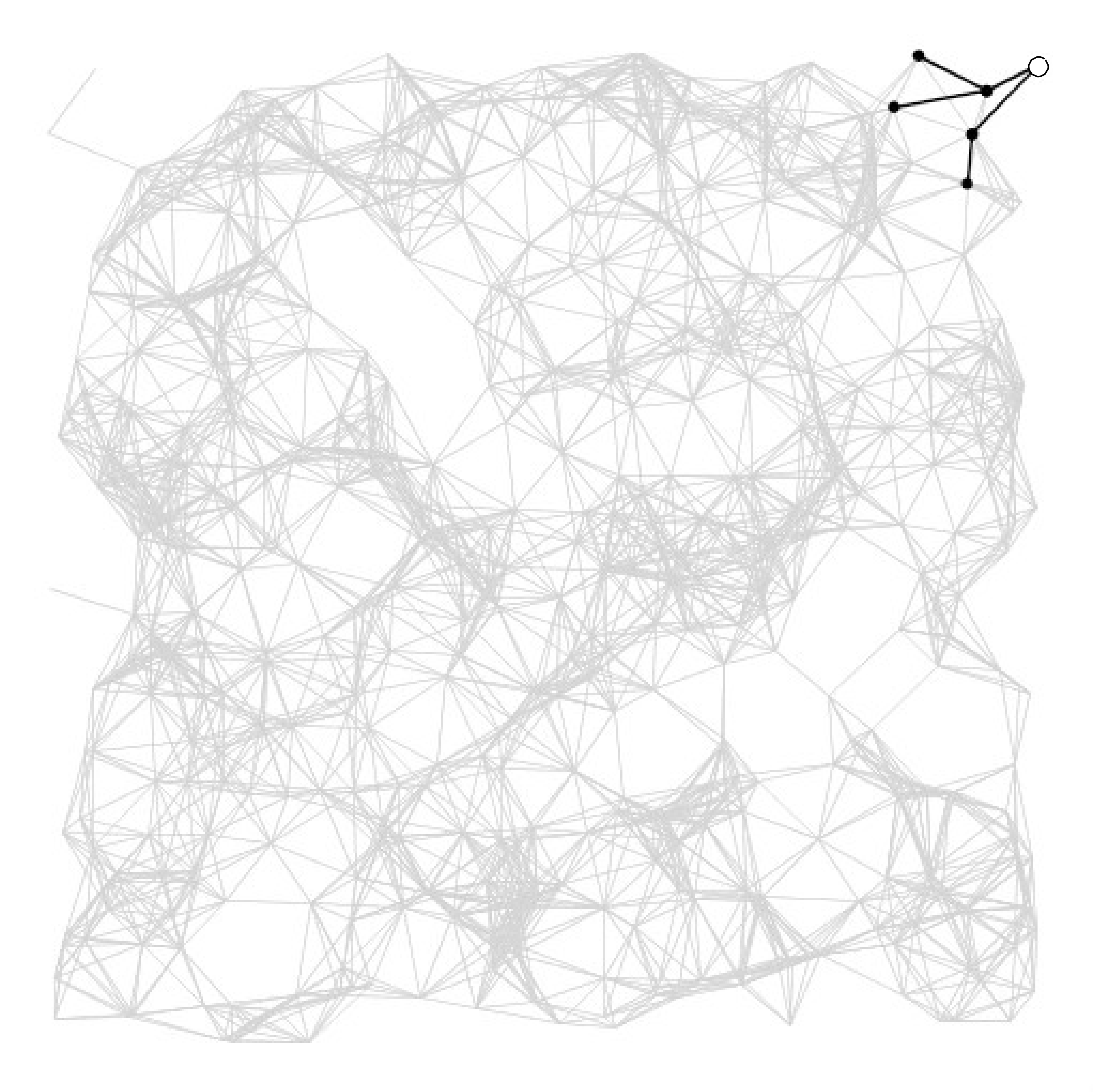
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

shortest-paths tree from vertex s

Dijkstra's algorithm visualization



Dijkstra's algorithm visualization



Dijkstra's algorithm: Java implementation

```
public class DijkstraSP
{
   private DirectedEdge[] edgeTo;
   private double[] distTo;
   private IndexMinPQ<Double> pq;

   public DijkstraSP(EdgeWeightedDigraph G, int s)
   {
      edgeTo = new DirectedEdge[G.V()];
      distTo = new double[G.V()];
      pq = new IndexMinPQ<Double>(G.V());

      for (int v = 0; v < G.V(); v++)
         distTo[v] = Double.POSITIVE_INFINITY;
      distTo[s] = 0.0;

      pq.insert(s, 0.0);
      while (!pq.isEmpty())
      {
          int v = pq.delMin();
          for (DirectedEdge e : G.adj(v))
             relax(e);
      }
    }
 }
```

relax vertices in order of distance from s

Dijkstra's algorithm: Java implementation

```
private void relax(DirectedEdge e)
{
   int v = e.from(), w = e.to();
   if (distTo[w] > distTo[v] + e.weight())
   {
       distTo[w] = distTo[v] + e.weight();
       edgeTo[w] = e;
       if (pq.contains(w)) pq.decreaseKey(w, distTo[w]);
       else                pq.insert     (w, distTo[w]);
   }
}
```

update PQ

Dijkstra's algorithm: which priority queue?

Depends on PQ implementation: V insert, V delete-min, E decrease-key.

PQ implementation	insert	delete-min	decrease-key	total
unordered array	1	V	1	V^2
binary heap	$\log V$	$\log V$	$\log V$	$E \log V$
d-way heap	$\log_d V$	$d \log_d V$	$\log_d V$	$E \log_{E/V} V$
Fibonacci heap	1 †	$\log V$ †	1 †	$E + V \log V$

† amortized

Bottom line.

- Array implementation optimal for dense graphs.
- Binary heap much faster for sparse graphs.
- 4-way heap worth the trouble in performance-critical situations.
-

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

4.4 SHORTEST PATHS

- *APIs*
- *shortest-paths properties*
- *Dijkstra's algorithm*
- *edge-weighted DAGs*
- *negative weights*

Shortest paths in edge-weighted DAGs

```
public class AcyclicSP
{
   private DirectedEdge[] edgeTo;
   private double[] distTo;

   public AcyclicSP(EdgeWeightedDigraph G, int s)
   {
      edgeTo = new DirectedEdge[G.V()];
      distTo = new double[G.V()];

      for (int v = 0; v < G.V(); v++)
         distTo[v] = Double.POSITIVE_INFINITY;
      distTo[s] = 0.0;

      Topological topological = new Topological(G);
      for (int v : topological.order())
         for (DirectedEdge e : G.adj(v))
            relax(e);
   }
}
```

topological order

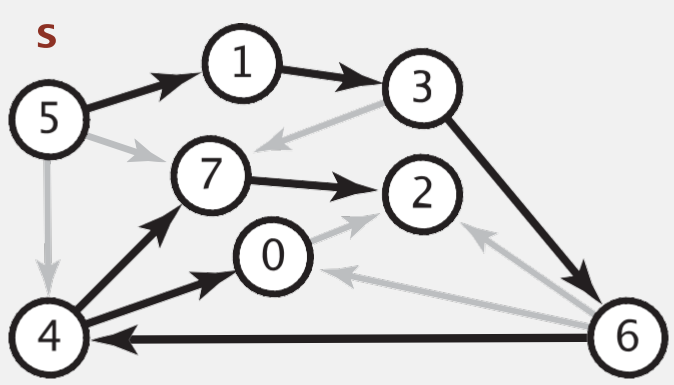
Longest paths in edge-weighted DAGs

Formulate as a shortest paths problem in edge-weighted DAGs.

- Neglizho peshat.
- Gjej shortest paths.
- Neglizho peshat në rezultat.

equivalent: reverse sense of equality in `relax()`

longest paths input		shortest paths input	
5->4	0.35	5->4	-0.35
4->7	0.37	4->7	-0.37
5->7	0.28	5->7	-0.28
5->1	0.32	5->1	-0.32
4->0	0.38	4->0	-0.38
0->2	0.26	0->2	-0.26
3->7	0.39	3->7	-0.39
1->3	0.29	1->3	-0.29
7->2	0.34	7->2	-0.34
6->2	0.40	6->2	-0.40
3->6	0.52	3->6	-0.52
6->0	0.58	6->0	-0.58
6->4	0.93	6->4	-0.93

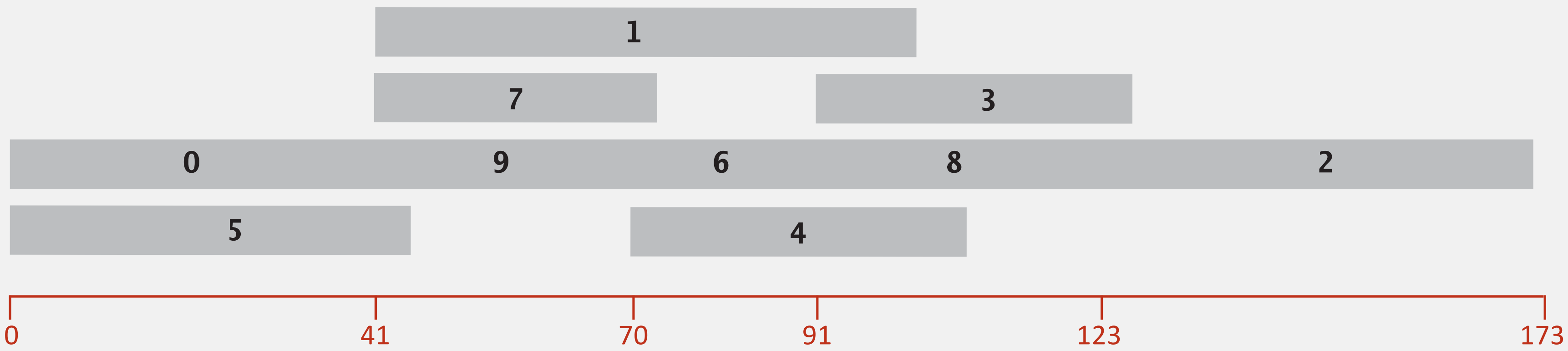


Key point. Topological sort algorithm works even with negative weights.

Longest paths in edge-weighted DAGs: aplikimi

Parallel job scheduling. Le të jetë një bashkësi e punëve me kohëgjatje dhe precedence constraints, schedule punët (duke gjetur fillimin për secilën prej tyre) ashtu që të arrihet koha minimale e përfundimit, duke I respektuar constraints.

job	duration	must complete before		
0	41.0	1	7	9
1	51.0	2		
2	50.0			
3	36.0			
4	38.0			
5	45.0			
6	21.0	3	8	
7	32.0	3	8	
8	32.0	2		
9	29.0	4	6	



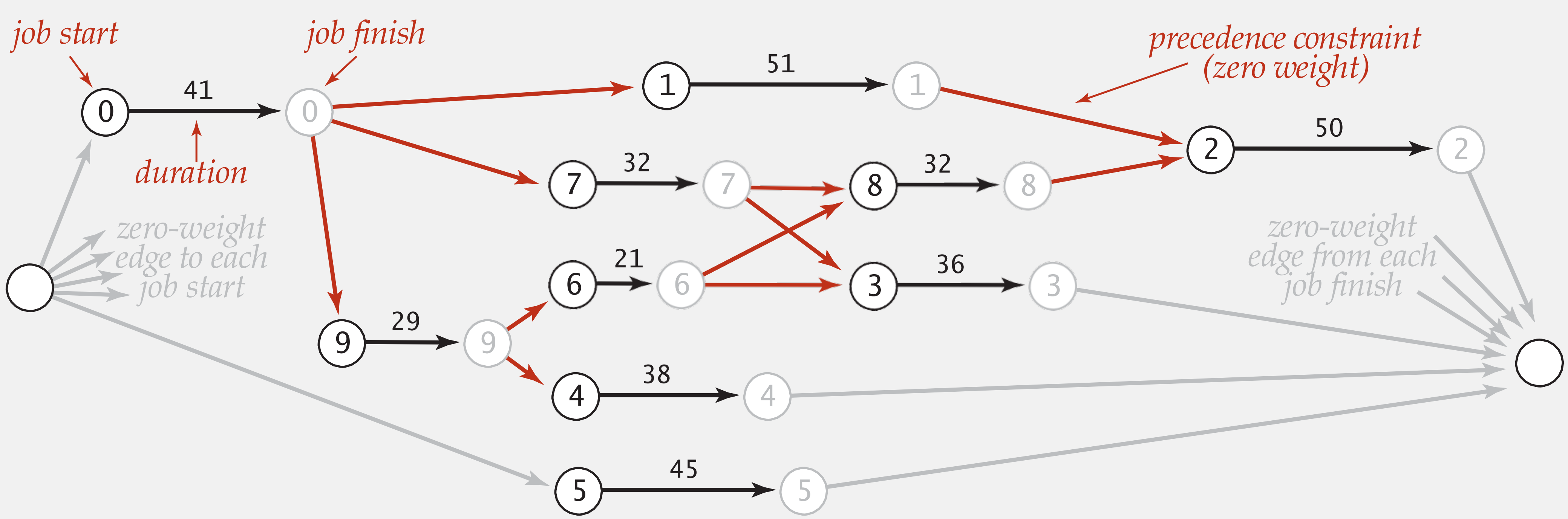
Parallel job scheduling solution

Critical path method

CPM. Për të zgjidhur parallel job-scheduling problem, krijo edge-weighted DAG:

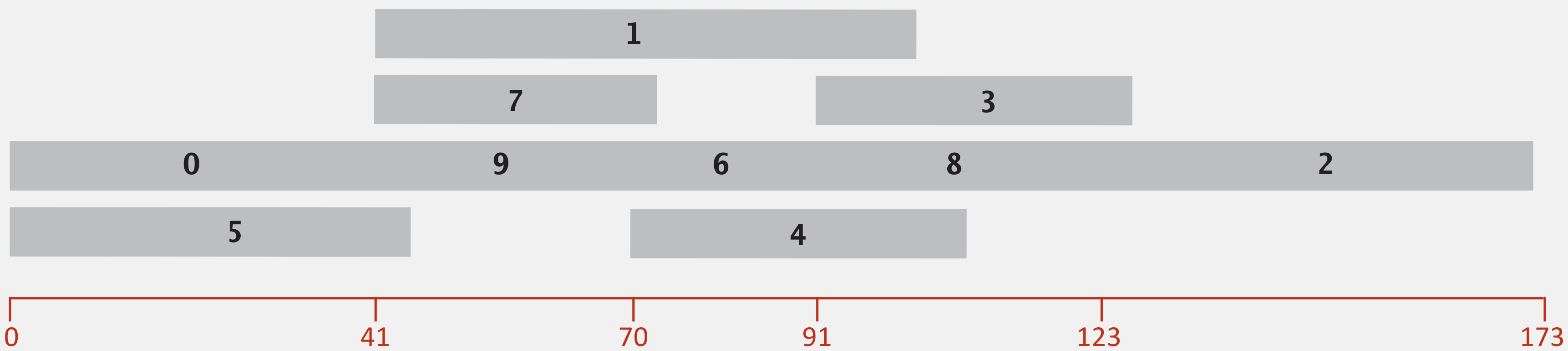
- Kulmi burimor dhe përfundimtar.
- Dy kulme (fillimi dhe fundi) për çdo punë.
- Tri segmente për çdo punë.
  - fillimi - fundi (peshuar me kohëzgjatje)
  - burimi - fillimi (0 peshë)
  - fundi - përfundimi (0 peshë)
- Një segment për çdoprecedence constraint (0 peshë).

job	duration	must complete before		
0	41.0	1	7	9
1	51.0	2		
2	50.0			
3	36.0			
4	38.0			
5	45.0			
6	21.0	3	8	
7	32.0	3	8	
8	32.0	2		
9	29.0	4	6	

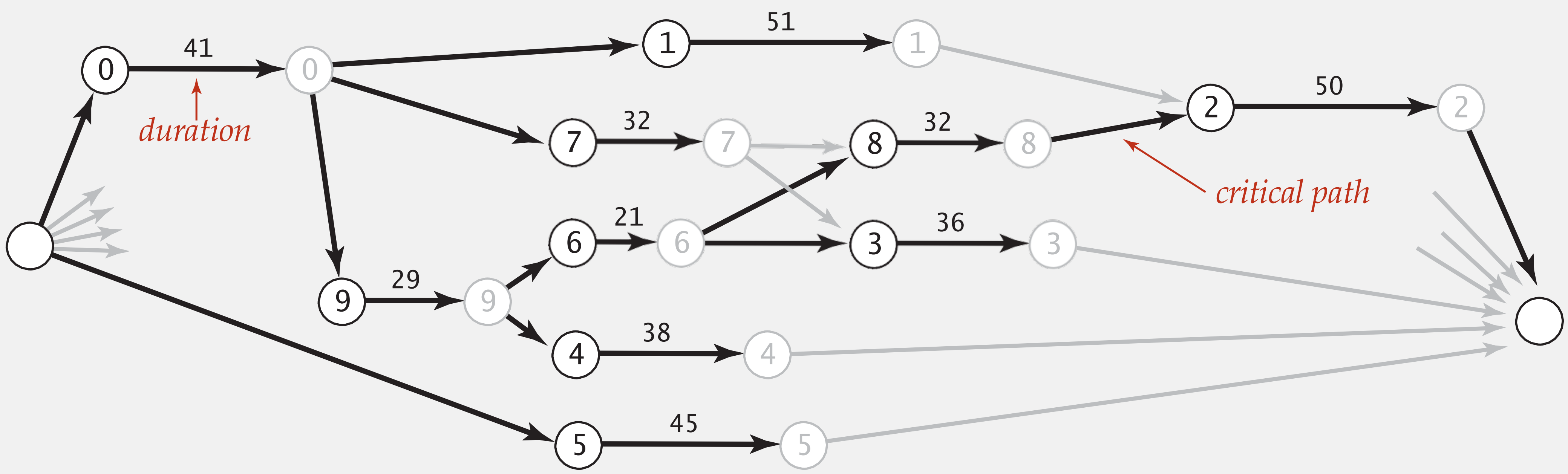


Critical path method

CPM. Use longest path from the source to schedule each job.



Parallel job scheduling solution



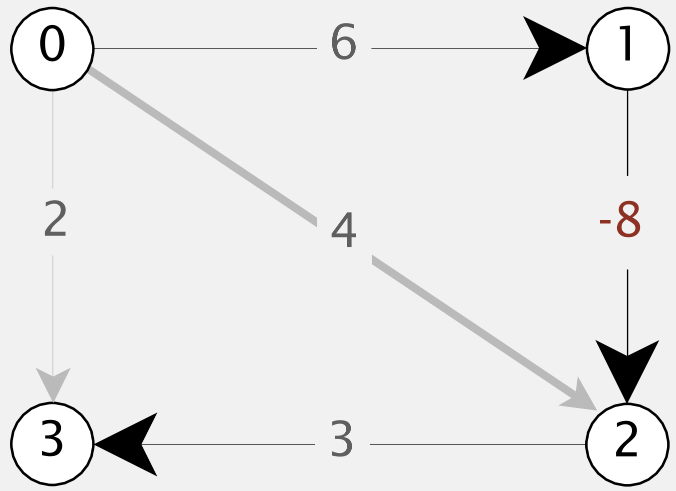
http://algs4.cs.princeton.edu

4.4 Shortest Paths

- APIs
- shortest-paths properties
- Dijkstra's algorithm
- edge-weighted DAGs
- *negative weights*

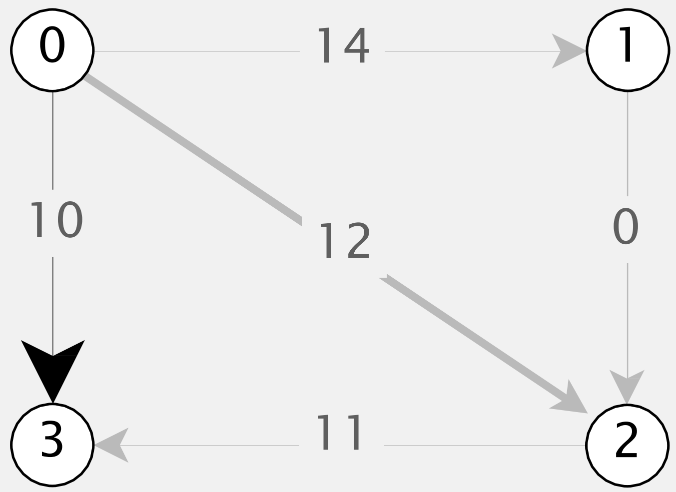
Shortest paths with negative weights: failed attempts

Dijkstra. Nuk funksionon me pesha negative të segmenteve.



Dijkstra pëzgjedh kulmin 3 menjëherë pas 0. Por, shortest path prej 0 tek 3 është 0→1→2→3.

Re-peshimi. Shtimi i një konstante secilit segment nuk funksionon.



Shtimi i peshës për 8 tek të gjitha segmentet ndërron shortest path prej 0→1→2→3 në 0→3.

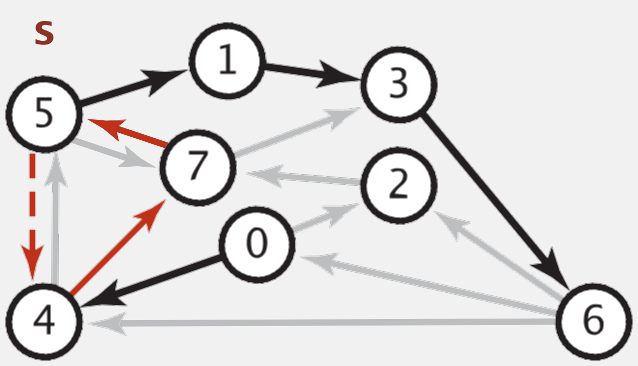
Përfundim. Duhet një algoritëm tjetër.

Negative cycles

Def. Një negative cycle është një directed cycle që ka shumën e peshës së segmenteve negative

digraph

4->5	0.35
5->4	-0.66
4->7	0.37
5->7	0.28
7->5	0.28
5->1	0.32
0->4	0.38
0->2	0.26
7->3	0.39
1->3	0.29
2->7	0.34
6->2	0.40
3->6	0.52
6->0	0.58
6->4	0.93



negative cycle (-0.66 + 0.37 + 0.28)

5->4->7->5

shortest path from 0 to 6

0->4->7->5->4->7->5...->1->3->6

Proposition. A SPT exists iff no negative cycles.

assuming all vertices reachable from s

Bellman-Ford algorithm

Bellman-Ford algorithm

Initialize distTo[s] = 0 and distTo[v] = ∞ for all other vertices.

Repeat V times:

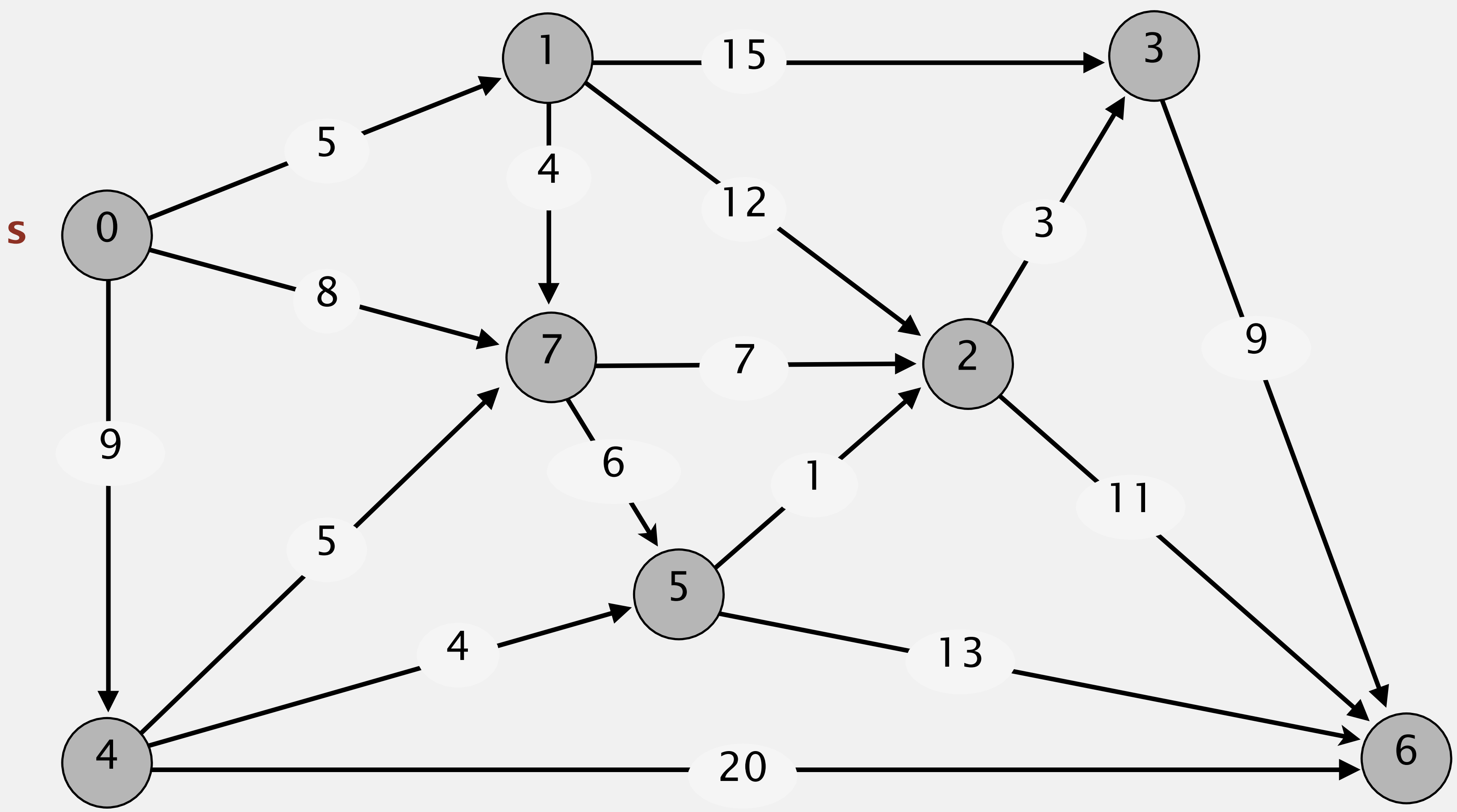
- **Relax each edge.**

```
for (int i = 0; i < G.V(); i++)
   for (int v = 0; v < G.V(); v++)
      for (DirectedEdge e : G.adj(v))
         relax(e);
```

pass i (relax each edge)

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.

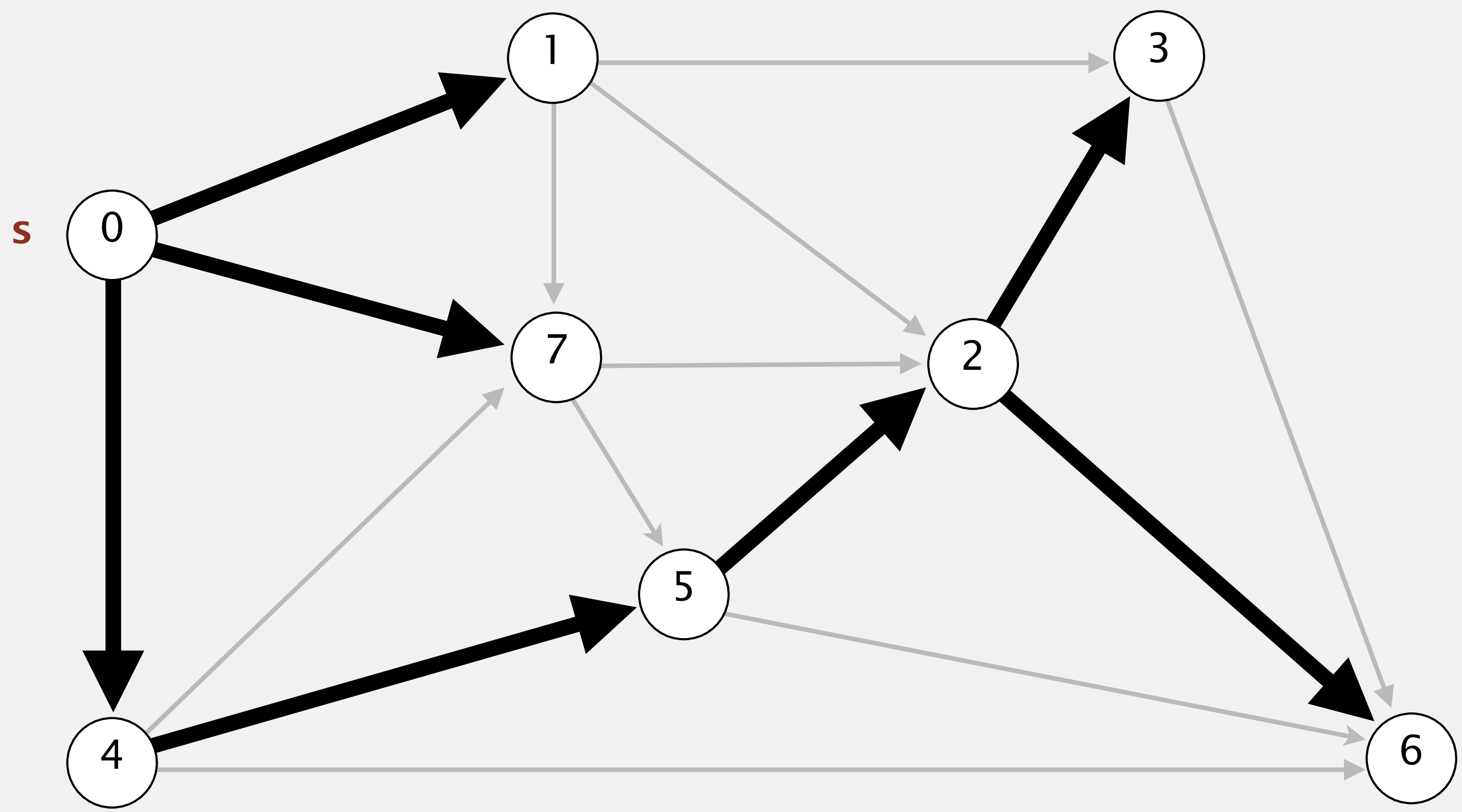


an edge-weighted digraph

0→1	5.0
0→4	9.0
0→7	8.0
1→2	12.0
1→3	15.0
1→7	4.0
2→3	3.0
2→6	11.0
3→6	9.0
4→5	4.0
4→6	20.0
4→7	5.0
5→2	1.0
5→6	13.0
7→5	6.0
7→2	7.0

Bellman-Ford algorithm demo

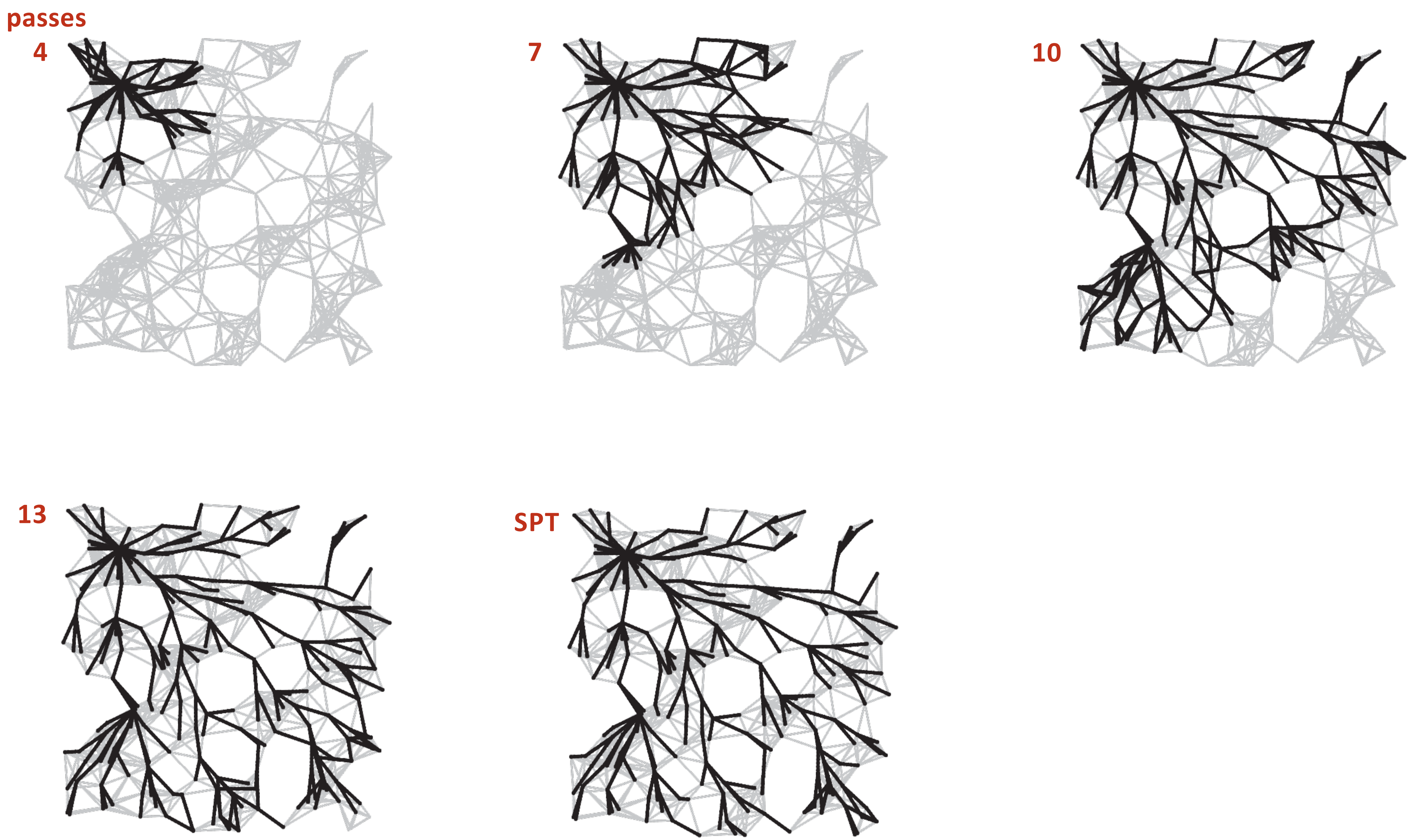
Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

shortest-paths tree from vertex s

Bellman-Ford algorithm: visualization



Bellman-Ford algorithm: analysis

Bellman-Ford algorithm

Initialize distTo[s] = 0 and distTo[v] = ∞ for all other vertices.

Repeat V times:

- **Relax each edge.**

Proposition. Dynamic programming algorithm computes SPT in any edge-weighted digraph with no negative cycles in time proportional to $E \times V$.

Pf idea. After pass `i`, found path that is at least as short as any shortest path containing `i` (or fewer) edges.

Bellman-Ford algorithm: practical improvement

Observation. If `distTo[v]` does not change during pass `i`, no need to relax any edge pointing from `v` in pass `i+1`.

FIFO implementation. Maintain queue of vertices whose `distTo[]` changed.

be careful to keep at most one copy of each vertex on queue (why?)

Overall effect.

- The running time is still proportional to $E \times V$ in worst case.
- But much faster than that in practice.

Single source shortest-paths implementation: cost summary

algorithm	restriction	typical case	worst case	extra space
topological sort	no directed cycles	$E + V$	$E + V$	V
Dijkstra (binary heap)	no negative weights	$E \log V$	$E \log V$	V
Bellman-Ford	no negative cycles	$E\ V$	$E\ V$	V
Bellman-Ford (queue-based)		$E + V$	$E\ V$	V

Remark 1. Directed cycles make the problem harder.

Remark 2. Negative weights make the problem harder.

Remark 3. Negative cycles makes the problem intractable.

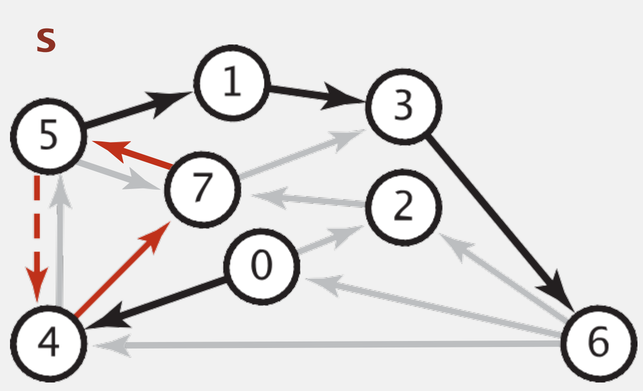
Finding a negative cycle

Negative cycle. Add two method to the API for SP.

`boolean`	`hasNegativeCycle()`	*is there a negative cycle?*
`Iterable <DirectedEdge>`	`negativeCycle()`	*negative cycle reachable from s*

digraph

```
4->5   0.35
5->4  -0.66
4->7   0.37
5->7   0.28
7->5   0.28
5->1   0.32
0->4   0.38
0->2   0.26
7->3   0.39
1->3   0.29
2->7   0.34
6->2   0.40
3->6   0.52
6->0   0.58
6->4   0.93
```

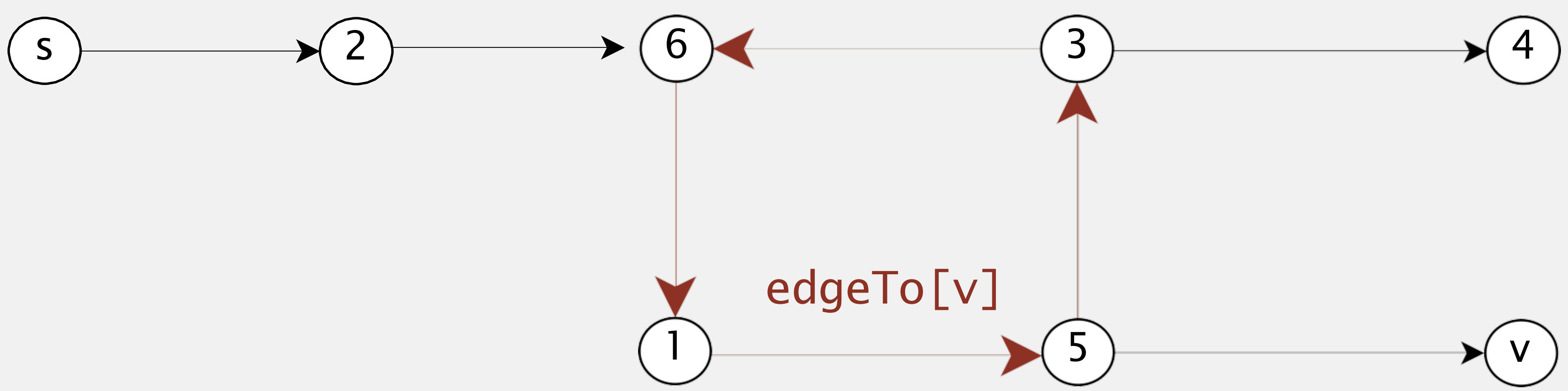


negative cycle (-0.66 + 0.37 + 0.28)

5->4->7->5

Finding a negative cycle

Observation. If there is a negative cycle, Bellman-Ford gets stuck in loop, updating `distTo[]` and `edgeTo[]` entries of vertices in the cycle.



Proposition. If any vertex `v` is updated in pass `V`, there exists a negative cycle (and can trace back `edgeTo[v]` entries to find it).

In practice. Check for negative cycles more frequently.